



**算法思想：**

将n个数放到n+1个桶，必然存在至少1个空桶，又由于每个桶的宽度相等，故最大间距必然不来自同一个桶。最大间距有两种情况，一种是落在相邻的两个区间，另一种是落在两个不相邻但中间区间（可能多个）没有元素的两个区间，并且是在前一个区间的最大值和后一个区间的最小值之间产生。

**算法步骤：**

1.读取文件，将数据放入num数组，并找出数组中最大值max和最小值min。O(n)复杂度

2.每个桶的宽度为length = (max - min) / (n - 1)。

3.试着将数放入桶中，其中num[i]应该放到下标为(num[i]-min)/length的桶里，每个桶只存储该桶内的最小值和最大值。 O(n)复杂度

4.遍历桶数组，找出前一个区间的最大值和后一个区间（若后一个区间内没有数字，则查找下一个区间）的最小值，相减得temp\_gap，然后与max\_gap值进行比较,记录大值，遍历结束即得max\_gap。 O(n)复杂度

C++实现如下：

#include <bits/stdc++.h>

using namespace std;

class Bucket

{

public:

    double max = DBL\_MIN;

    double min = DBL\_MAX;

};

int main()

{

    ifstream in\_file("D:\\algorithm\\max\_gap\\input.txt");

    ofstream out\_file("D:\\algorithm\\max\_gap\\output.txt");

    int n;

    in\_file >> n;

    double num[n];

    double max = DBL\_MIN;

    double min = DBL\_MAX;

    for (int i = 0; i < n; i++) //寻找输入最大值和最小值

    {

        in\_file >> num[i];

        if (num[i] > max)

        {

            max = num[i];

        }

        if (num[i] < min)

        {

            min = num[i];

        }

    }

    double length = (max - min) / (n - 1); //计算桶宽度

    Bucket bucket[n];                      //桶数组

    for (int i = 0; i < n; i++)

    {                                         //将输入的数字放入桶

        int target = (num[i] - min) / length; //目标桶号

        if (bucket[target].min > num[i])

            bucket[target].min = num[i];

        if (bucket[target].max < num[i])

            bucket[target].max = num[i];

    }

    double max\_gap = 0;          //最大间距

    double left = bucket[0].max; //保存区间左端点

    for (int i = 1; i < n; i++)

    {

        if (bucket[i].max > DBL\_MIN) //桶中有数，bucket[i].max被覆盖

        {

            double temp\_gap = bucket[i].min - left;

            if (temp\_gap > max\_gap)

                max\_gap = temp\_gap;

            left = bucket[i].max;

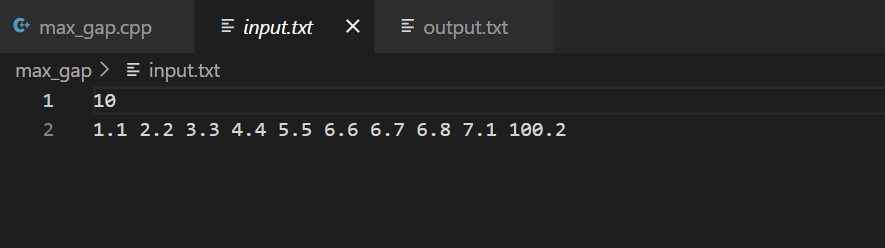
        }

    }

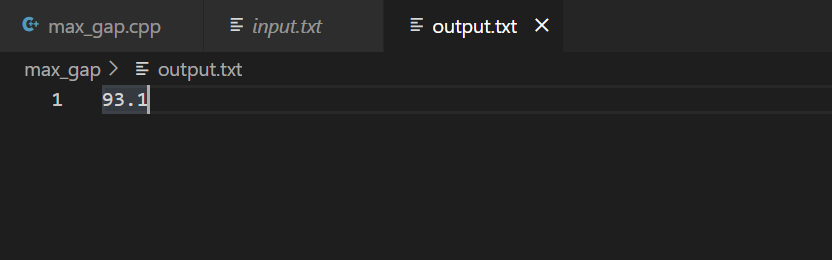
    out\_file << max\_gap;

}

input.txt



output.txt



命令行状态

